02 | 程序员也要关心整个系统和公司成本吗？

2019-12-02 庄振运

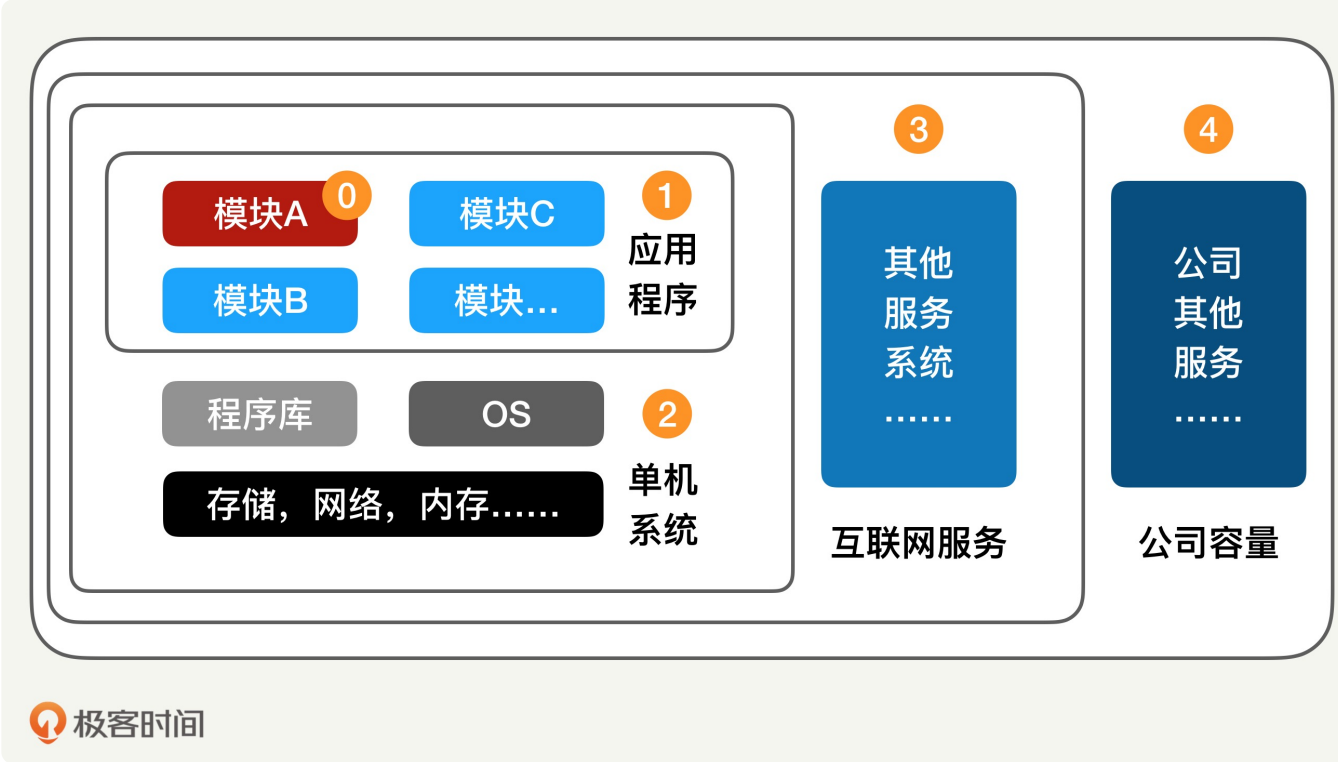
讲述：庄振运
时长 13:33　大小 18.64M

你好，我是庄振运。

上一讲我们谈了，作为一个程序员，你所负责的软件模块的性能是很重要的。如果写的程序性能不好，轻则通不过开发过程中的性能测试这一关，严重的话，还会为以后的业务生产环境埋下很多地雷和炸弹，随时会踩响和爆炸，从而影响公司的业务和运营。

代码性能的重要性，不仅仅局限于程序员所直接负责的软件模块，它对其他相关软件模块、模块所在的应用程序、单机系统的设计、互联网服务的质量、公司的运营成本，甚至对我们共同生活的地球都很重要。

这一讲，我们就来说说这几个方面。为了方便说清这几方面的关系，我画了下面这张图。



我来简单解释一下这张图：

首先，红色模块是我们负责的模块（标示 0），它和其他模块一起构成了整个应用程序（标示 1）；

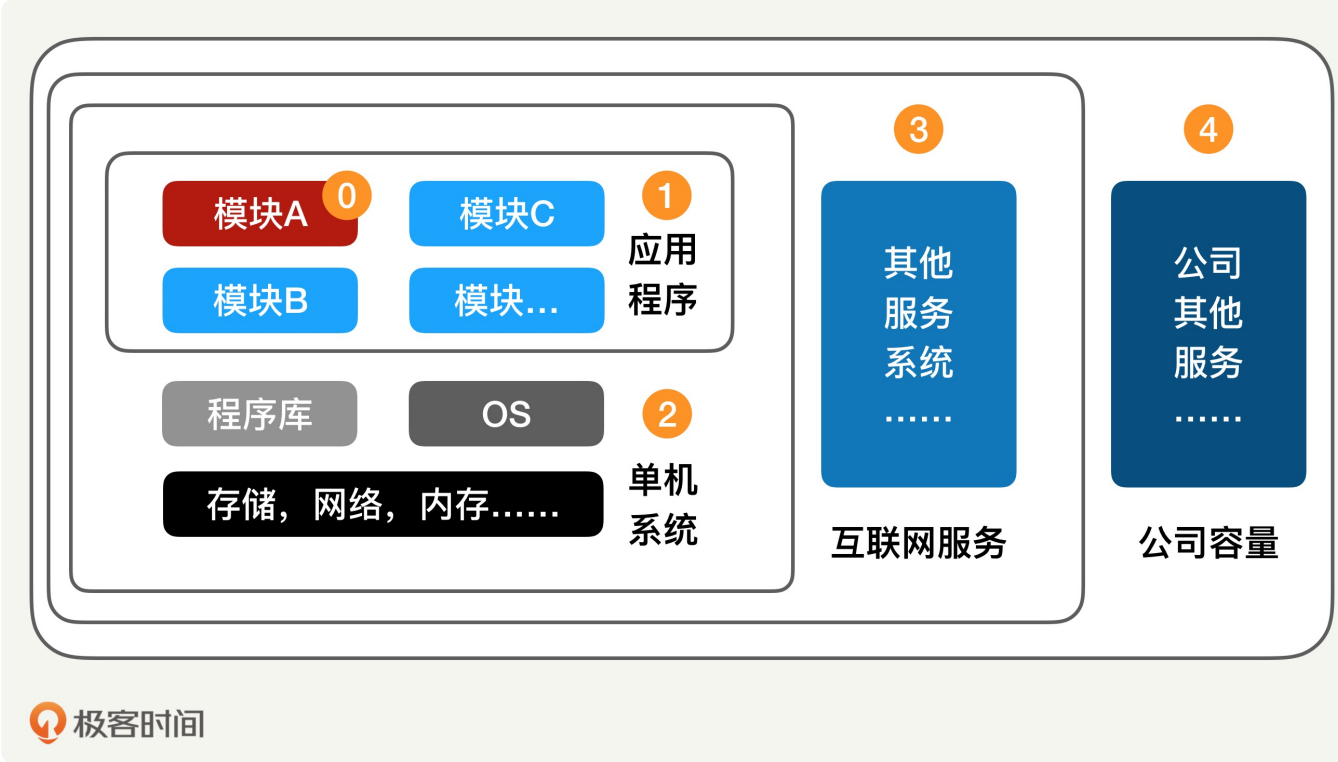
这个应用程序运行在服务器和 OS 上面，构成了一个单机系统（标示 2）；

几个单机系统一起组成一个互联网服务（标示 3），来面向客户；

这个服务和其他服务一起，需要公司的硬件容量支持，从而占用公司的商业成本（标示 4）；

最后，别忘了，我们共同生活在这个可爱的绿色星球上。

应用程序的性能（标示 1）



我们先从标示 0 和 1 开始，也就是模块和应用程序。

我们每个人负责的代码模块，一般都不是孤立存在的，都要和其他模块交互。模块之间是唇齿相依的。如果一个模块性能不好，一定会在某种情况下影响到其他模块，甚至是整个程序的性能和服务质量。唇亡齿寒的道理我们都懂，所以每个软件模块的性能都需要严格把关。

具体到我们自己的模块来说，或许在开发、测试的环境中，这个模块看起来运行正常，但等到了生产环境，一旦流量上去，性能不好的模块很快就会被曝光。尤其是在流量很高的时候，如果性能不佳，公司的运维同事一定会疲于奔命，更甚者会导致公司业务受损，这时性质就很严重了。

很多性能问题都会被根因分析。如果根因分析定位到是某人所负责的模块拖了大家的后腿，写这个模块的程序员不仅会被其他程序员鄙视，还可能会被老板找去“喝茶”。

回到性能问题上。性能问题可以表现在很多指标上，比如吞吐量（Throughput），服务延迟（Latency），可靠性（Reliability），延展性（Scalability）等等。根据模块所在的应用程序的性质，其中的一个或者几个指标会相对比较重要些。举个简单的例子，如果一个应用程序所在的互联网服务是面向终端客户的（比如微信用户），那么客户的服务延迟一定是极为重要的性能指标。

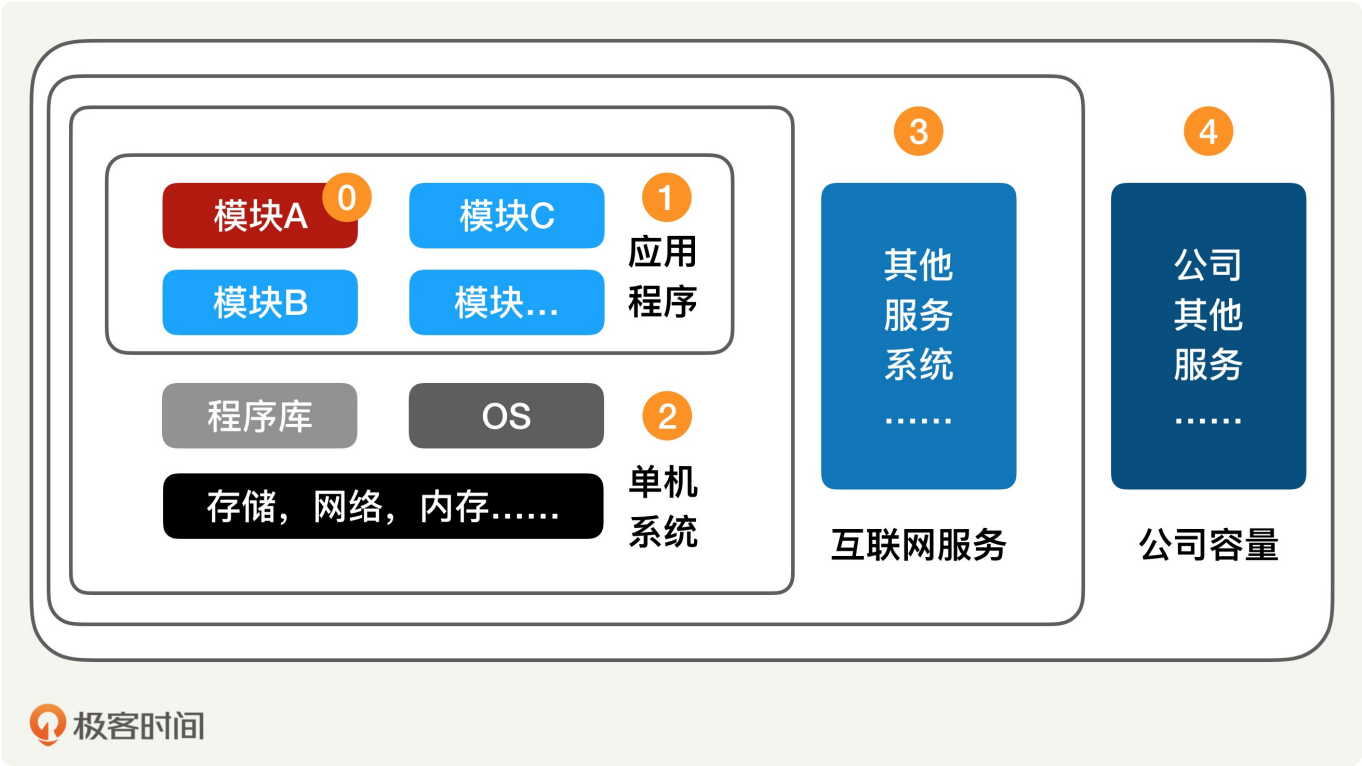
具体来说，假如端到端的服务延迟有最大的延迟允许，比如不能超过 2 秒钟，那么这个服务所需要的应用程序或者微服务，一般也都会有自己的最大延迟预算。假设这个端到端服务需要三个微服务或应用程序来串联，那么，每个应用程序都会分到一定的延迟预算，比如最大 1 秒。

同理，我们所负责的模块也会根据程序的逻辑设计分到相应的延迟预算，比如 300 毫秒，如下图红色模块所示。

这种情况下，如果我们的模块在流量适度变大时，处理时间超过 300 毫秒的预算，那这个模块的延展性显然就不够了，很可能会导致整个端到端服务的延迟超标。

单机系统的性能（标示 2）

讨论完了模块和程序，我们再看看单机系统。



我们的模块所在的应用程序（或者微服务），是运行在服务器的硬件和操作系统上面的。对这台服务器而言，这是个单机系统，包括软件和硬件的整个垂直全栈。现在的系统都非常复杂，软硬件之间的交互也复杂而微妙，并且随着各个构件的升级而经常变化。

单机系统的软硬件构件包括操作系统、程序库、存储系统、CPU、内存、还有网络等等。这些构件都会或多或少地影响上层程序的性能。

我举一个简单的例子，这个例子在后面的文章中还会仔细介绍。在我以前的公司，曾经有一个应用程序需要不断地把数据同步写到底层存储系统。这个应用程序后来被发现有性能问题，我们花了很多时间去做分析，后来终于找到原因。

这个性能问题的表现是：数据同步写入的延迟有时候会非常高。原因是**底层的存储系统同时服务好几个应用程序。**

由于底层存储系统有各方面的限制，当多个应用程序同时使用这个存储系统时，每个应用程序延迟方面的性能并不能得到保证，因此导致了某个应用程序的读写被严重推迟，并最终导致了后者的性能问题。

更进一步到设计层面来讲，从我们负责的模块和应用程序角度来看，**对下层的软硬件构件越是了解，就越有可能设计出性能优越的模块和应用程序。**

比如，很多数据存储方面的服务和应用程序在设计时，需要仔细考虑各种存储系统的技术趋势和性能特征。这些性能特征包括存储速度、价格、容量大小、易失性等。比如传统的 DRAM 内存就是一种存储，它速度很快，但是价格贵、容量小，并且所存数据不能长期保存，一断电数据就会丢失。

最近几年一种新的非易失性内存（NVM，Non-volatile Memory）的出现，打破了这一传统，数据可以长期保持，但是速度稍微慢一些。同样的，传统的硬盘存储容量大、价格低，但是速度最慢。最近几年固态硬盘（SSD） 的大量采用，在很多新设计的在线系统中已经作为标准配置，几乎取代了传统硬盘。

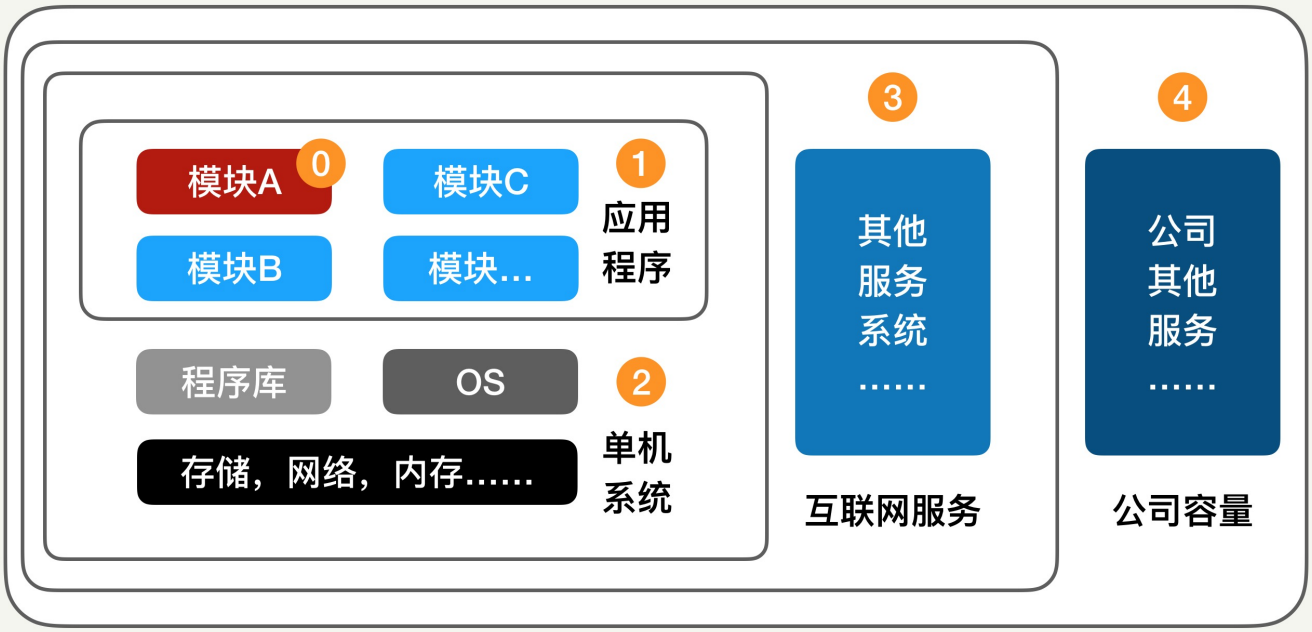
我们程序员作为自己模块甚至整个应用程序的设计者，如果能充分考虑这些硬件的性能特征和技术趋势，就可以设计出性能好、高效率的软件。

下面这张表格我列举了四种存储硬件，分别是传统 DRAM 内存、NVM 内存、硬盘和固态硬盘；并且比较了它们的五个指标。

存储硬件	价格	延迟	吞吐量	容量大小	数据失忆
传统DRAM内存	最高	最小	最高	最小	是
NVM内存	高	小	高	小	否
HDD硬盘	最低	最大	最低	最大	否
SSD固态硬盘	低	大	低	大	否

极客时间

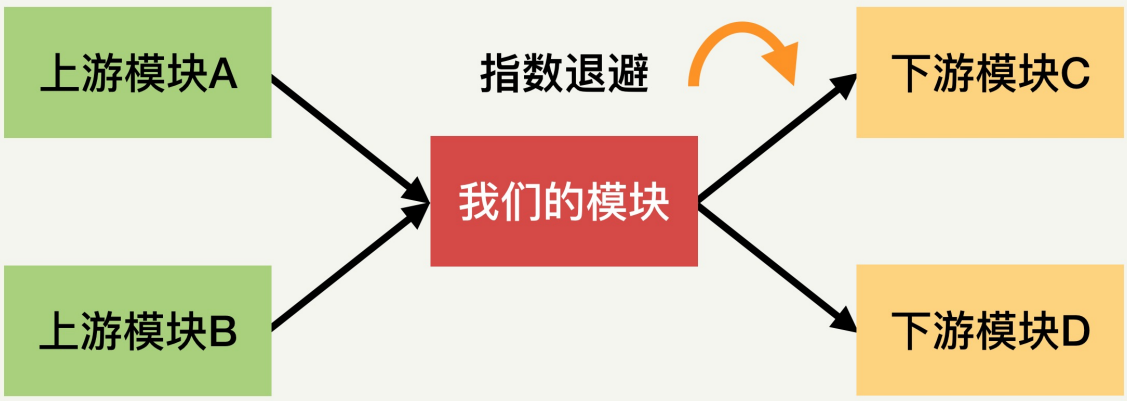
互联网服务的性能（标示 3）



极客时间

现代的互联网服务往往需要很多模块交互，并且客户流量会很大。我们所在的应用程序和系统经常只是整个互联网大服务的一部分，会有上游服务对我们产生请求，我们也会对下游服务发送请求。

比如下面的图示，我们所在的服务模块用红色标识，上游服务模块用绿色标识，下游服务模块用黄色标识。



极客时间

从公司运营的角度来看，整个互联网大服务的性能才是我们每个程序员真正关心和负责的。我们每人都需要从这个大局出发来考虑和分析问题，来设计自己的模块以及各种交互机制。否则，可能会出现我们的模块本身看起来设计得不错，但却对上下游模块造成不好的影响，进而影响整个大服务的性能。我来举一个真实的案例。

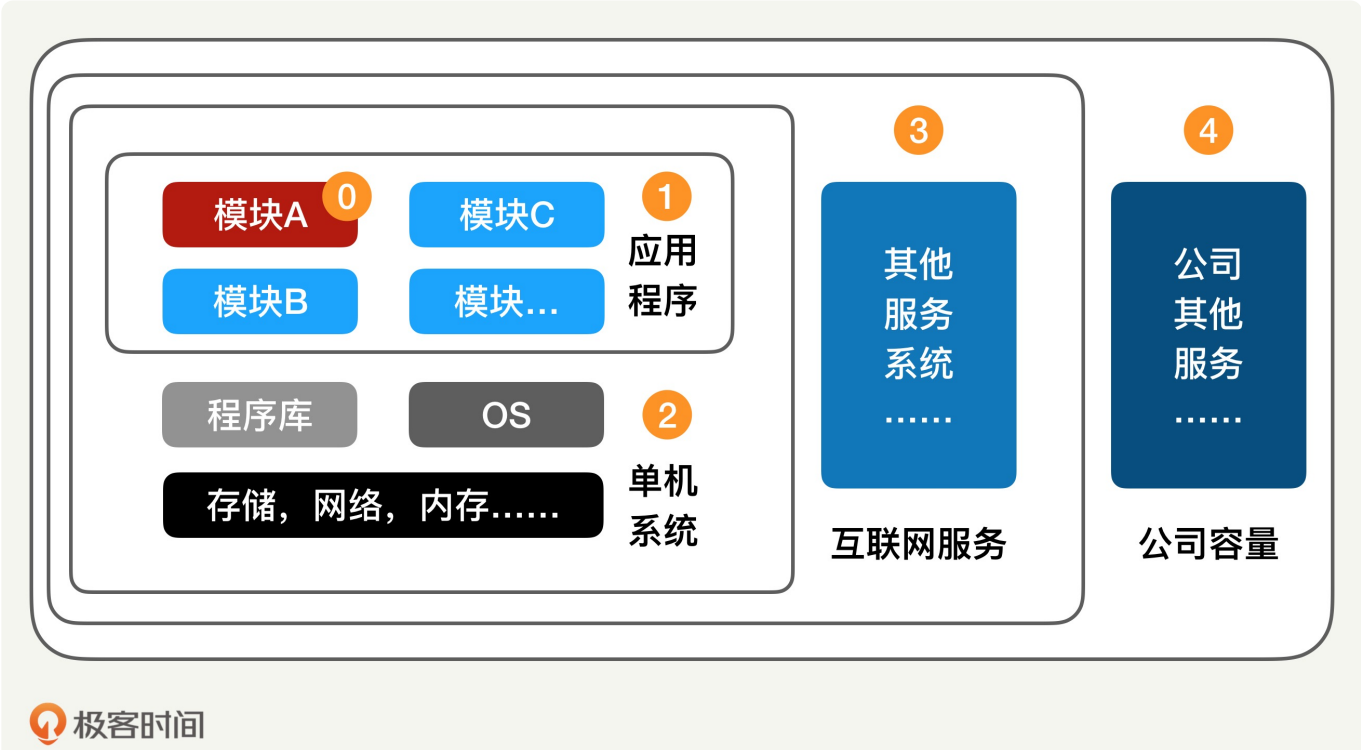
这个案例是从一次生产环境下的服务问题中发现的。某个下游模块出现延展性问题，服务的延迟变大，上游模块发出的请求排了很长的队。这个时候上游模块已经感觉到下游的性能问题，因为对下游请求的处理延迟已经大幅度增加了。

此时上游模块本应该怎么做呢？

它应该降低对下游模块的请求速度，从而减轻下游模块的负担。但是案例中的上游模块设计没有考虑到这一点。不但没有降低请求速度，反而发送了更多的请求，以求得更快的回答。这样无异于火上浇油，最后导致下游模块彻底挂掉，引发了整个服务的瘫痪。

后来我们学到的教训就是，串联的服务模块中，上游模块必须摒弃这样雪上加霜的服务异常尝试，应该采用**指数退避机制**（Exponential Backoff），通过快速地降低请求速度来帮助下游模块恢复（上游模块对下游资源进行重试请求的时间间隔，要随着失败次数的增加而指数加长）。

公司的成本（标示 4）



我们所负责的互联网服务的性能直接影响公司的成本。

一个高性能的服务，在服务同等数量的客户时，需要的成本会比较小。具体来说，如果我们的服务是计算密集型，那么就应该尽量优化算法和数据结构等方面来降低 CPU 的使用量，这样就可以用尽量少的服务器来完成同样的需求，从而降低公司的成本。

现如今是大数据时代，公司在服务器和数据中心以及网络等容量方面的支出是很可观的。尤其是大的公司比如脸书，腾讯等，公司有很多的数据中心和几百万台的服务器。如果公司的每个服务都做到高性能，替公司节省的运营成本是非常巨大的。

同时，面向互联网服务的容量规划和效率管理也很重要。如果能科学地管理容量，准确地预测未来需求，并逐步提升容量的效率，就能把公司这方面的成本管理和节省好，从而不至于浪费资金在不必要的多余容量上。

我们共同的绿色地球

最后，让我们跳出“我们的公司”这样的小格局，放眼全球，甚至我们人类的大格局。我们只有一个共同的地球，我们有责任让她保持绿色。

现在的时代，感谢互联网的发展和大数据时代的来临，全球各公司的数据中心已经在消耗大量的能源。从咱们国家来看，2018 年，国内的数据中心用掉的电量比整个上海市用电量还大，占全国用电量的 2.3%。全球来看也类似，数据中心在 2018 年消耗了全球 3% 以上的电量。这个耗电量已经是差不多整个英国全国用电量的两倍。更严重的是，这样的用电还在飞速增长，差不多每三年或四年就翻一倍！

所以，我们每个人，其实都负有责任来降低能源消耗。虽然生活中有多种方式可以降低能源消耗，我们的日常工作其实也是重要的一环。如果每个人能把负责的代码优化一下，服务高效一些，我们就是在拯救我们共同的地球，让她永葆绿色！

总结

对代码和程序的性能优化，以及对系统容量的效率提升，和我们共同关心爱护的东西息息相关。从代码模块，到整个系统，到互联网服务，到公司运营，再到我们的社会，都依赖于我们每个人的责任和贡献。

你和我或许是一介普通工程师和程序员，但人们常说“位卑未敢忘忧国”。我们虽然没必要拔高到忧国忧民的高度，但是也要认真做好我们的份内份外的事情。

思考题

回顾你最近接触过的软件模块或者正在写的代码，有没有和其他的系统模块有交互关系，它们之间是如何交互的？

如果你的模块性能不好（不管是响应时间很慢，还是发出过多请求），有没有可能对其他系统模块造成影响？这个影响会不会造成整个系统和服务的严重后果甚至瘫痪？

欢迎你在留言区分享自己的思考，与我和其他同学一起讨论，也欢迎你把文章分享给自己的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　01 | 程序员为什么要关心代码性能？

下一篇　03 | 导读：专栏是怎么设计的？需要哪些知识？

精选留言 (11)

写留言

西西弗与卡夫卡
2019-12-04

上游系统不做避让设计，异常情况下就是自己对自己发起了DDoS攻击。下游系统也要做熔断设计，以保护自己。

展开

作者回复: 没错；双重保护最好。

5

Middleware
2019-12-03

为了国家，为了地球，优化优化代码吧

展开

作者回复: 哈哈，说得没错哦. :)

5

glutton
2019-12-03

老师好，刚刚看到上新的这个课，就出手啦，目前正在做这方面的工作，特别缺少这种系统的讲解
1，我负责的系统，是整个公司服务链路中重要一环，目前全链路延时在200ms左右，允许给我们系统的最大时延是50ms（极端情况跨异地数据中心）
2，刚刚看到机械硬盘、SSD等对比，希望后续的课程，能详细列出成本、性能数据，以...

展开

作者回复: 谢谢留言和建议。我会尽量满足要求。学完后欢迎和我交流感想。SSD和硬盘的价格和性能其实一直在变（技术演化很快），所以只能给出大体对比，你可以Google搜索一下最新的价格以及每个厂商/型号的性能。如果有具体的问题（比如性能指标），我也可以看看帮你找找。

1

3

Liang Xu
2019-12-05

想到了Bufferbloat 显现. 如果很多人同时应用下游服务, 一遇到堵塞就开始减少请求, 这样就是所有人都减少. 一段时间之后是不是就产生周期性的算力需求? 学网络的时候学的

作者回复: 很好的思考。好的退避机制会引入随机数来决定行为，可以避免和极大地降低周期性的可能。

2

Q
2019-12-04

作为运维工程师，请问老师该怎么在性能工程领域帮助开发同事？ 实现自我的成长。希望老师能在接下来的一些课能具体谈谈这方面的东西。

展开

作者回复: 你是问开发和运维的有效配合吗？这方面有好几处：
-程序设计阶段，他们最好征询你们的意见，因为你们知道硬件，系统和容量；一般来收开发人员不太关心这些。
-程序测试阶段。你们要和他们密切交互，将“问题”扼杀在摇篮里。
-部署阶段。你们主导，但是需要经常让他们改进发现的问题。
-运行阶段。比如修Bug就是。

行下一首歌
2019-12-03

我看完就一句话深深地映入我脑海——多快好省。

展开

作者回复: 你历史一定学得很好。哈哈

Linuxer
2019-12-03

回顾你最近接触过的软件模块或者正在写的代码，有没有和其他的系统模块有交互关系，它们之间是如何交互的？
RPC、接口（其实RPC也算接口） 这两种方式

如果你的模块性能不好（不管是响应时间很慢，还是发出过多请求），有没有可能对其...

展开

Jxin
2019-12-03

1.我负责的系统模块是一个老项目的子系统。各服务间rpc调用超时均为6s。这会导致只要系统中任意一个子系统出现问题，由于这个6s等待的存在，就会导致所有关系方线程资源都被占用着。于是我调小了超时时间。然后快速失败却更加剧了故障系统的故障。最终我在入口处用mq销峰控制消费速率或则说任务并行度，借此保障系统稳定。然而，这依旧是有问题的，存在很大的浪费。因为如果瓶颈系统能支撑，单机能并行200的任务，而不能...

展开

作者回复: 恩，具体的系统真的事很复杂，要考虑各种因素。指数退让其实道理很直白易懂；实践起来也容易。基本道理就是如果不能及时获得下游回答，就假设下游很忙，那就降低请求速度（用指数方式降低，比如每次降低一半）。比如超时等待可以持续增加。

1

董飞
2019-12-03

老师，您好。我是做测试的，怎样往性能测试工程师发展？具体都要学什么？编程要学到什么程度？谢谢老师解答。

展开

作者回复: 这个专栏里面应该会有些答案帮助。

abctao
2019-12-02

期待后续讲解，谢谢！

展开

夜空中最亮的星（华仔...
2019-12-02

作为运维工程师 终于找到了下一个方向 性能工程师

展开

作者回复: :) 终于找到组织了！